Trust-Aware Peer Sampling: Performance and Privacy Tradeoffs

Davide Frey, Arnaud Jégou, Anne-Marie Kermarrec, Michel Raynal, Julien Stainer

Trust-Aware Peer Sampling: Performance and Privacy Tradeoffs

Davide Frey[a], Arnaud Jégou[a], Anne-Marie Kermarrec[a],
Michel Raynal[b,c], Julien Stainer[b]

[a] *INRIA-Rennes Bretagne Atlantique, Rennes, France*
[b] *IRISA, Université de Rennes 35042 Rennes Cedex, France*
[c] *Institut Universitaire de France*

Abstract

The ability to identify people that share one's own interests is one of the most interesting promises of the Web 2.0 driving user-centric applications such as recommendation systems or collaborative marketplaces. To be truly useful, however, information about other users also needs to be associated with some notion of trust. Consider a user wishing to sell a concert ticket. Not only must she find someone who is interested in the concert, but she must also make sure she can trust this person to pay for it.

This paper addresses the need for trust in user-centric applications by proposing two novel distributed protocols that combine interest-based connections between users with explicit links obtained from social networks à-la Facebook. Both protocols build trusted multi-hop paths between users in an explicit social network supporting the creation of semantic overlays backed up by social trust. The first protocol, *TAPS2*, extends our previous work on *TAPS* (Trust-Aware Peer Sampling), by improving the ability to locate trusted nodes. Yet, it remains vulnerable to attackers wishing to learn about trust values between arbitrary pairs of users. The second protocol, *PTAPS* (*Private TAPS*), improves *TAPS2* with provable privacy guarantees by preventing users from revealing their friendship links to users that are more than two hops away in the social network. In addition to proving this privacy property, we evaluate the performance of our protocols through event-based simulations, showing significant improvements over the state of the art.

1. Introduction

The advent of Online Social Networks (OSN) has shifted the core of Internet applications from devices to users. Explicit social networks like *Facebook*, or *LinkedIn* enable people to exploit real-world connections in an online setting. Collaborative tagging applications such as *delicious*, *CiteULike*, or *flickr* form

[☆] A preliminary version of this paper appeared in [1].

dynamic implicit networks of users on the basis of their online activities, interest profiles, or search queries. Users can not only access and introduce new available content but they become themselves accessible through the online infrastructure.

Explicit vs implicit networks. Existing online social networks can be grouped into two main categories: explicit and implicit. In explicit networks, users explicitly determine which other users they should be connected to. In *Facebook* or *MySpace*, they issue and accept friendship requests. In *Twitter*, they decide that they wish to follow the tweets of specific users. The topology of the resulting network reflects the choices of users and often consists of links that already exist between real people. Explicit networks are thus very useful to exploit existing connections but provide little support for discovering new content [2, 3].

Implicit networks fill this gap by taking an approach which allows users to discover new content, and acquaintances [4]. They form dynamic communities by collecting information about collateral activities of users, such as browsing websites or tagging documents, URLs, or pictures. A given user may or may not be aware of the other members of her own communities. Other users should be clearly visible if the purpose of the application is to discover new people, while they may be hidden for the sake of privacy when they are simply being used as proxies to access new interesting content. In either case, the ability to establish new social connections is key to identifying new and useful data.

The problem addressed in this paper: trust. Recent years have seen the emergence of a significant number of research efforts and applications exploring the power of the explicit and implicit paradigms. Nonetheless, a lot more can be achieved if both approaches are combined into a single framework. Consider the following example. John, who lives in London, bought two electronic tickets for a classical-music concert in Paris, Berenice by Handel, but an unexpected event makes him and his friend unable to travel to Paris. The concert is tomorrow and John would like to sell the tickets to someone who can actually attend the event. Unfortunately, while John has many friends interested in classical music, they are all based in the United Kingdom. He does know a few people in Paris, but they are mostly people he met while traveling and who do not share his musical tastes. He tries calling a few of them but his best bet is Joseph, who claims to have a friend whose parents often go to classical-music concerts. Unfortunately, this friend of Joseph is out of town and Joseph does not know how to reach his parents. As a last resort, John posts a message on a French classical music forum, linking to an EBay ad. However, none of the classical music fans on the forum are responsive enough and some of them even become suspicious that the electronic ticket being sold by this new forum user is actually a fake.

Previous work. In our previous work [1], we introduced Social Market, a system that could solve John's problem by combining the information obtained from implicit and explicit social networks. Social Market consists of the combination of two gossip-based layers: a trust-aware peer sampling, and a clustering protocol. The former, *TAPS* [1] (Trust-Aware Peer Sampling), provides each node

(user), n, with a continuously changing sample of the network, consisting of a set of other nodes n'. For each n', it identifies a path along which n can reach it using the links of the explicit social network and associates it with an inferred trust value obtained from the product of the trust values of the links in the path. The clustering protocol incorporates this information into a similarity-based implicit social network. This provides each node with information about which nodes provide the best compromise between trust and profile similarity.

In our example, Social Market allows John to identify someone who, albeit not knowing him directly, is interested in the concert and trusts him enough to buy an electronic ticket from him. Specifically, interest similarity allows John to identify François, a music teacher from Paris who is trying to buy two tickets for one of his students and himself. The explicit social network instead highlights that François is actually the cousin of a French colleague of John's wife. This allows the two to gain confidence in each other and thus complete a safe transaction without external help. Despite these interesting features, the existing versions of Social Market and of its *TAPS* [1] protocol are vulnerable to attackers wishing to learn about the trust values between given pairs of users. This is particularly problematic as an application like Social Market can only be successful in the presence of a large enough number of users. Attackers attempting to learn private information such as trust may in fact discourage users from participating into the system or from providing accurate trust values.

Content of the paper. In this paper, we address this issue by presenting two novel trust-aware peer sampling protocols, *TAPS2* and *PTAPS* (*Private TAPS*), that improve *TAPS* [1] on several aspects. First, they incorporate an optimized view exchange strategy that leads to a more efficient exploration of the network. Second they bias the peer-sampling process to favor connections between nodes that have high-trust relationship. Third, they introduce a trust threshold that discourages, in the case of *TAPS2*, or forbids, in the case of *PTAPS*, communication between nodes that have poor mutual trust values. Finally, *PTAPS* incorporates additional mechanisms designed to provide provable privacy guarantees. Our two new protocols identify different trade-offs in the design space of trust-aware peer sampling. *TAPS2* seeks to provide the clustering protocol with the most trustworthy nodes. *PTAPS* strikes a balance between this goal and the need to hide trust values from third parties.

We analyze both protocols in terms of their properties and their performance by combining theoretical considerations with extensive simulations. Specifically, we show that our protocols compare favorably with existing solutions by providing results that are even closer to those obtained by protocols equipped with global system knowledge. This makes *TAPS2* and *PTAPS* directly applicable to situations like the social transaction example described above. Moreover, our results open new directions for making existing gossip-based applications more robust in the presence of unreliable users.

2. System Model and Background

We consider a system consisting of a set of users equipped with interconnected computing devices that enable them to exchange information in the form of messages. Each user is associated with a *user profile* that characterizes her interests, her past behavior, her geographical location, and whatever other information the user wishes to add. A profile is a vector of strings that can represent, for example, URLs, words, or phrases. We refer to each such string as a *keyword*. Each *keyword* in a profile is also associated with a counter, its *weight*, which counts how many times the keyword has been added to the profile. The weight basically measures how relevant a given keyword is with respect to the other keywords in the profile. Keywords can be added by the user, or they can be extracted from her browsing history, as well as from her interaction with the system. To simplify the notation, we refer to a user and her profile with the same symbol $u \in U$, where U is the universe of all user profiles. We also use the terms *node* and *user* interchangeably to refer to a user and the machine associated with her. Profiles can be compared with each other using a standard similarity metric. Even though the protocols we describe in this paper can operate using any such metric, we concentrate on the well known *cosine similarity* [5], which measures normalized overlap. More precisely we consider

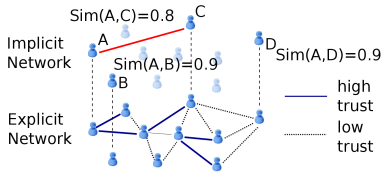
$$s_{u1,u2} = cos(u_1, u_2) = \frac{u_1 \cdot u_2}{||u_1|| \cdot ||u_2||}$$

where u_1 and u_2 are profile vectors, the scalar product $u_1 \cdot u_2$ is the sum of the products of the scores associated with corresponding items in u_1 and u_2, and $||u_1||$ is the norm of vector u_1.

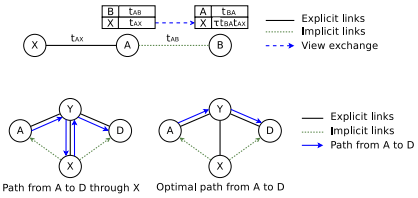
2.1. Social Market

Social Market (SM) [1] is a novel distributed application enabling users to identify previously unknown social acquaintances that, at the same time, are similar to them and can be trusted through a chain of explicit social connections, the *trusted path*. Selecting similar users is crucial when searching for the right people for a given transaction, but also when building recommendation or data-dissemination systems. Trust enables the implementation of transactions without external help and increases users' confidence in recommendation results. Users interact with Social Market by proposing items that they wish to exchange with other users. An item can be, for example, an object to sell, an object to buy, but it can also be a question that is being asked and that needs an answer. When a user u creates an item, she associates it with an item profile, similar to what is done in [6]. Structurally, an item profile is identical to a user profile. Upon creation, the system initializes the item profile to the corresponding user profile. The user then completes the creation by selecting which keywords from this profile clone should be kept, which should be removed, and which, if any, new keywords should be added.

In the example of Section 1, John creates an item for the Handel concert in Paris. Prompted with a profile that contains, among other things: *computer*



(a) Importance of a trusted path: A selects C, rather than B or D as a neighbor even if it has a lower similarity value because it is reachable through a trusted path (high-trust links).



(b) Exchange of trust information (top) and Short circuiting of trusted paths (bottom).

Figure 1: Trusted paths.

science, *cycling*, *mountaineering*, *violin*, *rock*, and *classical music*, John decides to keep only *violin* and *classical music* in the item profile. He then adds two more keywords, *Paris* and *Handel*, and decides to keep only the latter in his user profile as he's not interested in being notified about other items associated with Paris. Once an item has been created, the goal of Social Market is to lead this item to *meet* other users who (i) are interested in the item, (ii) can be trusted and can trust the creator of the item, and (iii) can be reached through a *trusted path* on a social network (Figure 1a).

To make this possible, SM users can create explicit social links. While similar to friendship links in systems like Facebook, SM links also have an additional feature: *trust*. Upon establishing a link, users declare how much they trust each other by agreeing on a value in $(0, 1]$. This associates a symmetrical trust value with each friendship link, resulting in an undirected social graph with arc weights between 0 and 1. The corresponding values are similar to the degrees of friendship/confidence specified in some existing social networks such as CouchSurfing[1]. In particular, if user A assigns a value close to 0 to the link to a user B, it does not necessarily mean that A completely distrusts B, but it may simply mean that A does not know B enough to express a positive opinion.

2.2. Social-Market Architecture

As presented in [1], SM consists of two layers of gossip protocols: *TAPS* (Trust Aware Peer Sampling) and *TAC* (Trust-Aware Clustering). *TAPS* provides each node with a sample of the network containing references to other nodes and associated trust values. *TAC* uses this information to select the nodes that offer the best compromise between trust and interest similarity as shown in Figure 1a. These protocols operate on three data structures: the EXPLICIT view, the TAPS view, and the TAC view. The EXPLICIT view, directly

[1] www.couchsurfing.org is a social community supporting the exchange of accommodation between its users.

updated by the user, contains the node's explicit friends. The TAPS view, updated by *TAPS*, contains a continuously changing set of random nodes. Finally, the TAC view is updated by *TAC* to contain the set of nodes offering the best trade-off between trust and similarity.

Each entry in any of these three views contains information about a target node: its IP address and port, its user profile, a timestamp indicating when the information in the entry was generated, as well as a *trusted path* and an *inferred trust value*. The *trusted path* is a path leading to the target node and consists of a chain of other nodes connected by explicit social links. The *inferred trust value* is an estimate of the trust that the owner of the view can have in the target node, and is computed as a combination of the trust values in the *trusted path* between them.

Each edge in a trusted path carries some amount of uncertainty about the trustworthiness of one endpoint with respect to the other even if all the nodes in the path fully trust each other. To model this, we define the *inferred trust* of a path as the product of the trust values of its edges, weighted by a *trust transitivity* coefficient, τ, expressing how much a node values other nodes' recommendations. Given a path $u_1, u_2..., u_n$, let $t_{i,j}$ be the trust value between users u_i and u_j, then the inferred trust between u_1 and u_n is the following.

$$\tilde{t}_{1,n} = \tau^{n-2} \prod_{i=1}^{i=n-1} t_{i,i+1} \tag{1}$$

The trust transitivity factor, τ, is applied once per recommended link; so for a path involving n nodes, τ is applied $n-2$ times. Lower τ values cause trust to decay faster with path length. With $\tau = 0.7$ trust decays from 1 to 0.49 in only three hops. Finally, it is worth observing that in the special case of the EXPLICIT view, the *trusted path* only contains the target node, and the *inferred trust value* is the real trust value toward this node.

2.2.1. Trust-aware peer sampling

TAPS follows the general structure of a peer-sampling protocol [7] and populates the TAPS view with an ever-changing set of references to other nodes. When a node enters the system, the *TAPS* layer initializes its TAPS view by inserting one entry for each of the node's explicit neighbors. During the course of the protocol, these entries are exchanged with entries received from other nodes. Initially nodes providing new entries will be the node's explicit friends, then the node's friends' friends, and so on. Periodically, each node contacts the node with the oldest timestamp[2] from either its TAPS view or its EXPLICIT view, and the two exchange a random half of their respective TAPS views (if the target is from the TAPS view), or of the union of their TAPS and EXPLICIT views (if the target is from the EXPLICIT view). For each entry, a node exchanges

[2]The choice of the node with the oldest timestamp facilitates the removal of departed nodes as described in [7].

information about a path leading to it and the corresponding trust. Ultimately, this causes the entries in a node's TAPS view to include a continuously changing set of nodes with the associated inferred trust values.

Trust Inference in TAPS. When exchanging views, nodes update the inferred trust values towards the exchanged nodes according to Equation (1). Moreover, they achieve this by exchanging only aggregate trust values and not the trust values of each link in the path. This makes it possible to protect trust information from a passive adversary that is curious to learn about trust values, but that does not actively combine the values of multiple paths to infer those associated with explicit links. As an example, consider a node A exchanging information with another node B as shown in the top diagram of Figure 1b. A sends B a subset of its view as well as the path it has to reach B and the value it has for $t_{A,B}$, the inferred trust between A and B. B uses this trust information to update the inferred trust values before adding the nodes to its own view. Specifically, let $p_{A,X}$ be the path from A to X, and $t_{A,X}$ be the associated inferred trust value, then B computes its own path and trust for X, as follows. First it verifies if it already has a value for $t_{A,B}$. If so, it keeps the highest value between the received one and its own. It then uses the selected $t_{A,B}$, and compute the resulting path $p_{B,X}$ and trust value $t_{B,X}$ as

$$p_{B,X} = p_{B,A} \cdot p_{A,X} \text{ and } t_{B,X} = \tau t_{B,A} t_{A,X}$$

where $\cdot$ denotes path concatenation. Similarly, if B already had a path to X that was better than the XAB path computed above (not shown in the figure), it would attempt to use it to build a new path to A going through X, and would keep it if $\tau t_{B,X} t_{X,A} > t_{B,A}$.

Dealing with multiple references to the same node. As views evolve, a node inevitably receives multiple references for the same node. In the presence of multiple trust values for the same node, X, a node, A, always selects the largest. Two references for the same node may also contain slight differences in profiles. A combines the highest trust values with the most recent profile information.

Even if a node opts for the highest trust value when choosing between two trusted paths for the same node, the trust update mechanism tends to favor the creation of longer and longer paths, possibly containing loops. For a node A to be able to choose the shorter of two trusted paths to the same destination, A must receive the longer path when the shorter one is still in its view. This, however, is made unlikely by the fact that the contents of a node's view keep changing under the effect of the peer-sampling protocol.

Short-Circuiting Trusted Paths. To limit the proliferation of longer and longer paths, *TAPS* [1] includes a short-circuiting mechanism [1]. Consider the situation depicted in the bottom diagram of Figure 1b. Node X holds a reference to D with a trust value $t_{X,D}$ and a path going through Y and sends it to A. Node A should combine this with the trust value it has for node X, $t_{A,X}$, also

obtained through Y. However, this would lead to a path that uselessly goes twice through node Y and once through X.

Node A can prevent this by short-circuiting the path thereby also improving its trust value. Specifically, A knows that the aggregated impact on trust of the path segment YX cannot be greater than τ^n, n being the number of useless links in the path, each link being counted once for each time it is traversed (n=2 in this case). It can therefore conclude that the trust value of node D as seen from A, $t_{A,D}$ is at least the following.

$$t_{A,D} \geq \frac{t_{A,X} \cdot t_{X,D}}{\tau^n}$$

2.2.2. Trust-aware clustering

The TAPS view constitutes the main source of information for SM's upper layer: its trust-aware clustering protocol (*TAC*). *TAC* is a variation of well-known protocols [8, 4] and maintains a TAC view that collects the nodes that offer the best compromise between trust and similarity. Upon receipt of another node's view, a node X combines the received view and its own TAC view, and selects the entries associated with the best trade-off between similarity and trust.

Score computation. Using the similarity and the trust it has toward a node N, a node X can compute a score $\sigma_{X,N}$ as a weighted product between its trust and its similarity: $\sigma_{X,N} = {s_{X,N}}^{2-\epsilon} {t_{X,N}}^{\epsilon}$, $\epsilon \in [0, 2]$, where ϵ, the *trust weight*, determines the importance of trust in the trade-off, $t_{X,N}$ denotes the trust between X and N and $s_{X,N}$ is the similarity between X and N.

TAC view exchanges. Similar to *TAPS* [1], each node periodically selects the node with the oldest timestamp in its TAC view and exchanges the content of its TAC view with it. These further exchanges speed up convergence with respect to only relying on information from the *TAPS* layer by providing nodes that tend to have higher similarities.

2.2.3. Trust verification

A final aspect of Social Market is its trust-verification mechanism. While not an architectural component, this mechanism plays a major role in guaranteeing safe transactions based on the information provided by *TAPS* and *TAC*. Specifically, trust verification allows nodes to obtain a confirmation for the trust values of the entries that have remained in their TAC views for at least c cycles ($c = 5$ as in [4]). This provides protection from nodes that attempt to cheat on their trust values when communicating with nodes that are not direct friends. Consider Figure 1b (bottom): node D could try to enter A's cluster by making A believe that it has a high-trust link to node Y. To prevent this, A asks D to forward a *verification message* back to A along the trusted path. The message starts with a trust value of 1. Nodes along the path multiply the message's value by τ and by their trust for the node they received the message from. Y thus multiplies 1 by $\tau t_{Y,D}$ in the example. This process causes the *verification message* to reach A with the correct value for $t_{A,D}$ thereby invalidating D's cheating attempts.

2.2.4. Modifications of the explicit view

When two users declare themselves as friends in the social network, both nodes add an entry corresponding to their new friend in their EXPLICIT view. Once this is done, the new friends will start exchanging their views like with any of their explicit friends, and no further actions are needed. In the case of a friendship link removal, the nodes simply remove their old friend from their EXPLICIT view, and remove any path through this friend from their TAPS and TAC views. That way, they will stop exchanging entries from their explicit views, and all the paths containing this link will eventually be discarded when verification messages sent to check these paths will abort.

The remainder of this paper presents novel contributions to the Social-Market architecture in the form of two new *TAPS* protocols. The first protocol, *TAPS2*, described in Section 3, features a number of techniques that lead to significant performance improvements with respect to [1] along with a slightly lower risk of unauthorized access to trust information. The second, *PTAPS*, presented in Section 4, trades off a fraction of the improved performance to provide strong privacy guarantees.

3. Trust-Aware Peer Sampling (*TAPS2*)

TAPS2 improves the performance of the first version of *TAPS* by discouraging the creation of trusted paths with too-low trust values, while limiting the dissemination of trust information to distant areas of the network using three new mechanisms. First, it biases view selection to favor the choice of nodes associated with high trust values, improving view quality and reducing the likelihood that a non-trusted distant node may learn a lot about the local topology of the network. Second, it establishes a dynamic trust threshold to avoid communicating with nodes whose trust values are too low to be useful. Finally, it includes an improved view exchange mechanism leading to better trusted paths.

3.1. Biased View Selection

The first novel technique in *TAPS2* is a biased view-selection mechanism. In *TAPS* [1], a node that needs to integrate its view with a set of entries received from another node operates as in a standard peer-sampling protocol. It extracts a random subset of the union of its own and the received entries and keeps it as its new view. *TAPS2* seeks to improve view quality and provide better privacy for trust values by biasing the selection of a node's neighbors towards nodes associated with higher trust values.

Consider a node a that needs to update its view V_a with information received from another node V_r. First, a computes the union of the two views $V_a \cup V_r$. Then it selects n_{TAPS} nodes from $V_a \cup V_r$ one after the other by associating each candidate node with a probability obtained by dividing its trust values by the sum of all the trust values of the nodes in $V_a \cup V_r$ that have not been selected. Therefore, the probability that a node be in the selected sample increases with its trust value. This causes the view to be biased towards nodes with higher

values, leading to two important consequences. First, it reduces the likelihood that a node with a low trust value may enter the TAPS view. Second, it yields a more clustered topology making it less likely for a non-friend with a low trust value to gather information about the trust between a node and its friends.

3.2. Minimum-Trust Threshold

The second mechanism employed by *TAPS2* to limit the dissemination of trust information and the proliferation of long paths is a dynamic threshold on trust values. For any node n, this is defined as the minimum trust value that a hypothetical node n' with similarity 1 would need to enter n's cluster. Let σ_i denote the score of a node i from the CLUSTER view of n, and let t_i and s_i be respectively their reciprocal trust and similarity values. Because of the definition of score ($\forall i : \sigma_i = t_i^{\epsilon} s_i^{2-\epsilon}$), such a perfectly similar node would have a score of $\sigma_{n'} = t_{n'}^{\epsilon}$. For it to enter the CLUSTER view of n, this score would need to be higher than the lowest score of a node currently in the view, i.e. $t_{n'}^{\epsilon} > \min\{\sigma_i, i \in \text{CLUSTER}\}$, which is equivalent to $t_{n'} > \min\{\sigma_i, i \in \text{CLUSTER}\}^{1/\epsilon}$. Clearly, this minimum requirement also applies to nodes that have lower similarity values as their scores are necessarily lower than those of perfectly similar nodes with the same trust values. Thus, because the goal of the TAPS view is to feed the corresponding TAC view, *TAPS2* ignores all the nodes that have trust values lower than $\min\{\sigma_i, i \in \text{CLUSTER}\}^{1/\epsilon}$.

This threshold is dynamic: a node recomputes it whenever it modifies its cluster view. It then removes from its *TAPS2* view all the nodes whose trust values are lower than the threshold, and refuses the insertion in the view of similarly untrustworthy nodes. This process gets more and more refined as nodes select the candidates with the best scores as their neighbors. This causes a corresponding increase in the value of the threshold, making nodes more and more demanding as their TAC views improve.

Finally, it is worth observing that the trust threshold induces a maximal length for acceptable paths that can be calculated by: $d_{\max} = \log_{\tau}(t_{\min})$. Ultimately, this causes the threshold to have a positive impact on the quality of the TAPS and TAC views because it allows nodes to focus on the parts of the explicit network in which they can find the best neighbors.

3.3. Continuous path bootstrapping

The final improvement we introduce in *TAPS2* is a modification of the view exchange mechanism. In [1], each node initializes its TAPS view with its explicit neighbors. However, when communicating with nodes selected from TAPS view, it always exchanges information about nodes extracted from the same TAPS view without reintegrating fresh information about its explicit neighbors. With *TAPS2*, we improve this behavior by perpetuating the bootstrap process. At each cycle, a node A chooses a communication partner, N, with an equal probability from either its TAPS or its EXPLICIT view as in *TAPS* [1]. However, different from [1], it selects each of the entries to send to N with an equal probability from either of the two views. This leads to four types of exchanges: TAPS-TAPS, TAPS-EXPLICIT, EXPLICIT-EXPLICIT, and EXPLICIT-TAPS,

as opposed to the only three types TAPS-TAPS, TAPS-EXPLICIT, and EXPLICIT-EXPLICIT available in [1]. The additional exchange type (EXPLICIT-TAPS) causes information from the EXPLICIT view to enter the views of neighbors from the TAPS view, thereby making it easier to obtain shorter trusted paths.

4. Privacy-preserving Trust-Aware Peer Sampling (*PTAPS*)

TAPS2 assumes an honest-but-curious adversary that is also completely passive. Specifically, the adversary should not actively store information about trusted paths in order to calculate direct trust values between arbitrary nodes. *TAPS2*, in fact, hides the trust values of the individual segments composing a trusted path by disclosing only an aggregate trust value. As a consequence, an adversary, A, that was able to obtain information on a trusted path ABX with value t_X and on another $ABXY$ with value t_Y could easily compute the trust value of the link XY, as $t_{X,Y} = \frac{t_Y}{\tau t_X}$.

To understand the impact of this kind of attack, we ran simulations (we provide details on our experiments in Section 5.1) in which nodes store the information they receive and use it to infer the trust values associated with explicit links. An average node is able to discover the trust values of 40% of the 14000 explicit links in the social network in as little as 1000 cycles.

In this section, we address this vulnerability of *TAPS2* with *PTAPS*, a protocol that trades off part of *TAPS2*'s ability to discover the best paths for provable privacy guarantees even with active attackers. We present the details of the protocol in Section 4.1, and then prove its privacy properties in Section 4.2.

4.1. Protocol Description

PTAPS augments *TAPS2* with three novel features. First, it reduces the amount of information exchanged, to provide provable privacy guarantees. Second, it hides the intermediary links of a path from users, thereby preventing the attack described above. Finally, it strengthens the trust threshold to prevent all communication with untrusted users.

4.1.1. Non-disclosure of explicit friends.

The goal of *PTAPS* is to prevent an attacker from inferring the existence of an explicit link between arbitrary nodes outside its social circle, that is, between nodes that are at least two hops away from the attacker. The first step in this direction is to make it impossible to tell which of the paths and trust values advertised in a view update correspond to a node's explicit friends. An attacker that can identify a node's explicit friends can in fact use their trust values to infer those associated with the node's friends' friends, and so on.

To avoid this problem, *PTAPS* modifies the exchange mechanism of *TAPS2* by allowing a node to have entries corresponding to indirect paths to its explicit friends in its TAPS view. This allows nodes to learn alternative longer paths to their explicit friends, thereby hiding their status of friends from other nodes. Consider a node n with direct neighbors $N(n)$ that is performing a *PTAPS*

exchange with another node p. As in *TAPS2*, n chooses p either from its direct neighbors or from its TAPS view with equal probability ($P(p \in N(n)) = P(p \in V_{\text{TAPS}}) = \frac{1}{2}$). If $p \notin N(n)$ (p is not an explicit friend), then n selects the entries to send to p only from its TAPS view and not from its EXPLICIT view. This achieves the goal of hiding the explicit-friend status of the entries sent to non-friends during gossip exchanges in two ways. First, it replaces one-hop paths with longer paths making paths to explicit friends indistinguishable from the others. Second, it causes the probability of sending a *TAPS* entry about an explicit neighbor to be comparable to that associated with other nodes, making it difficult to guess the identity of friends based on statistical attacks.

However, nodes use this modified exchange mechanism only when exchanging information with non-friends. It is easy to see that a node must know the identities of its friends' friends in order to bootstrap the process. As a result, if $p \in N(n)$ (p is an explicit friend) n continues to operate as in *TAPS2* and chooses a view to send to p by selecting each entry with equal probability from the TAPS or the explicit view.

4.1.2. Trusted-path protection

Being unable to identify another node's friends is essential to protect trust information, but it is impossible if attackers have access to complete path information. As a result *PTAPS* also employs a novel *path-encryption* and routing mechanism. Specifically, it guarantees that if a node is represented in clear text in a path, then its predecessor and its successor appear in encrypted form. Moreover, a given node in a path can only decrypt information about its own predecessor and successor. This provides each node with the ability to manipulate and combine paths to other nodes, while hiding the information regarding the internal links of the path. For simplicity, we start by describing a basic version of path encryption that only hides the links of a path, and then modify it so that it also hides path length. Finally, we present the details of the path encryption protocol.

Basic path encryption. Essentially, an encrypted path consists of a sequence of routing blocks, each of them containing information about the next or the previous hop in the path. These blocks can either be *secret*, i.e. the identity of the node they refer to is encrypted, or *public*, i.e. the node they refer to is unencrypted and thus publicly known. A *secret* routing block, F_N, consists of two sub-blocks: an identity block containing F's identity, and a randomly generated block. The concatenation of the two sub-blocks is encrypted by a key private to N. Because of this, N is the only node that can decrypt the block and thus know that it refers to F. Also, the presence of a random-noise sub-block means that two routing blocks encrypted by the same node N and referring to the same node F will look different. We use the notations F_N and F'_N when we need to emphasize the fact that two blocks F_N and F'_N, while containing the same data, use a different random block. A *public* routing block F on the other hand simply contains F's identity, which is not hidden or encrypted by any key.

(01) **Periodically**
(02) **for all** $F \in ExplicitView$
(03) $F_N = \mathsf{encrypt}(F)$
(04) $F'_N = \mathsf{encrypt}(F)$
(05) send $Update(F_N, F'_N)$ to F

(06) **When** $Update(F_N, F'_N)$ received by F from N
(07) $N_F = \mathsf{encrypt}(N)$
(08) $N'_F = \mathsf{encrypt}(N)$
(09) $ExplView[N].Secret \leftarrow N_F \cdot F_N$
(10) $ExplView[N].Pub_{out} \leftarrow N \cdot F'_N$
(11) $ExplView[N].Pub_{in} \leftarrow N'_F \cdot F$
(12) send $Reply(N_F, N'_F)$ to N

(13) **When** $Reply(N_F, N'_F)$ received by N from F
(14) $ExplView[F].Secret \leftarrow F_N \cdot N_F$
(15) $ExplView[F].Pub_{out} \leftarrow F \cdot N'_F$
(16) $ExplView[F].Pub_{in} \leftarrow F'_N \cdot N$

Algorithm 1: Maintaining Routing Information.

Since we need to be able to follow an encrypted path both forwards and backwards, routing blocks always appear in pairs representing links. Specifically, a link $N - F$ contains a routing block F or $\mathrm{F_N}$ for node F, followed by a routing block N or $\mathrm{N_F}$ for node N, yielding for example $\mathrm{F_N N_F}$. The reason for reversing the order of N and F in the link representation (e.g. $\mathrm{F_N N_F}$) is to facilitate the routing process. Specifically, when reading the link from left to right, the first of the two blocks will be used by node N to identify F as a next hop. The second block will instead used by F to identify N as a previous hop. Reversing the link ($\mathrm{N_F F_N}$) enables navigation in the opposite direction, F using N as its next hop, and N using F as its previous one.

This model allows us to represent any path as a sequence of links. For example, the path consisting of the sequence of nodes A,B,C, and D can be represented by concatenating the links $A - B$, $B - C$, and $C - D$, yielding $\mathrm{B_A A \cdot C_B B_C \cdot D C_D}$. Note that the blocks referring to the first and last node are *public* as they identify the endpoints of the path.

Those corresponding to the first and the last node are not the only blocks in a path that may be *public*. Blocks corresponding to intermediary nodes may also be *public* as a result of the way the path was constructed. Nonetheless, as stated above, the goal of the protocol is to prevent an attacker, X, from discovering the explicit friends of any other node, N, that is at least 2 hops away. This leads to two requirements. First (i), a *secret* routing block referring to a node N must never be used together with a *public* routing block referring to the same node N. Second (ii), a path must never concatenate the *public* routing blocks corresponding to two neighboring nodes. Rather if a node in a path is represented by a *public* routing block, then both the previous and the following nodes in the path must appear as *secret* blocks. In this respect, it is

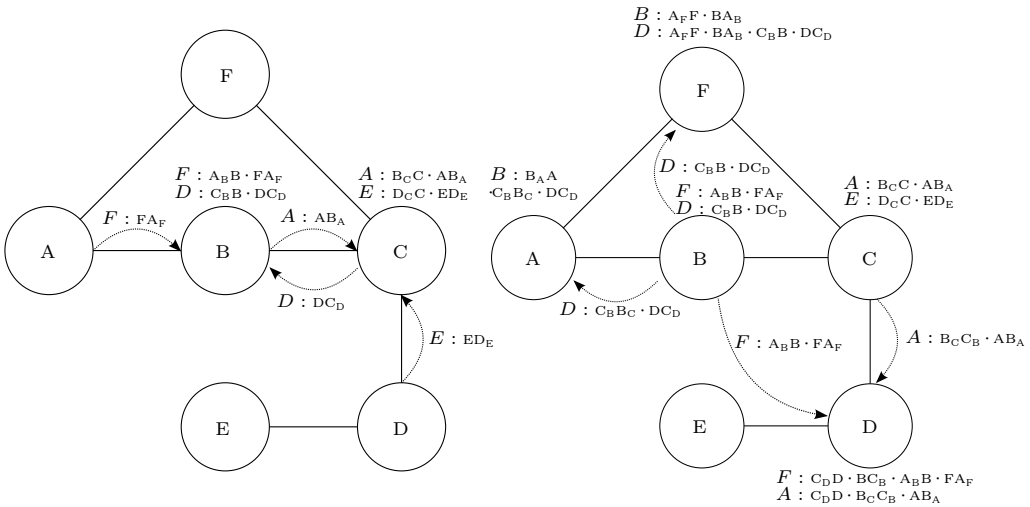
```
(01) Before sending a path P_AZ to an explicit friend C
(02)     if Z is an explicit friend of A
(03)         send ExplView[Z].Pub_out to C
(04)     else
(05)         Given B the node after A in P_AZ
(06)         P_BZ ← RemoveFirstLink(P_AZ)
(07)         send ExplView[B].Secret · P_BZ to C

(08) Before sending a path P_AZ to an Taps node C
(09)     Nothing special to do
(10)         send P_AZ to C

(11) When a path P_BA is received by a node A from a node B with P_BA.Dest = A
(12)     if P_AB ∉ Tapsview
(13)         P_AB ← ReversePath(P_BA)
(14)     else
(15)         if P_AB.trust < P_BA.trust
(16)             P_AB ← ReversePath(P_BA)

(17) When a path P_BC is received by a node A from an explicit friend B
(18)     // The path P_BC starts by a link where B is hidden
(19)     P_AC ← ExplView[B].Pub_in · P_BC

(20) When a path P_BC is received by a node A from an non-friend B
(21)     // The path P_BC starts by a link where B is public
(22)     P_AC ← P_AB · P_BC
```

Algorithm 2: Path management during view exchanges.

worth observing that the relevant order is that of the nodes in the path and not that of the blocks in the path representation.

As an example, consider the path consisting of the sequence of nodes F, A, B, C, and D in Figure 2b. The representation $\mathrm{A_F F} \cdot \mathrm{BA_B} \cdot \mathrm{C_B B} \cdot \mathrm{DC_D}$ is allowed (note that the node preceding B is A and not F; likewise the one following B is C and not D), while $\mathrm{A_F F} \cdot \mathrm{BA_B} \cdot \mathrm{C_B B_C} \cdot \mathrm{DC_D}$ is not. The first one only reveals that F and B share a friend, and that so do B and D, but it does not give any information about the identities of these common friends. However, the second representation, because of the concatenation of the blocks $\mathrm{BA_B}$ and $\mathrm{C_B B_C}$, reveals that B is the first node of the link $\mathrm{C_B B_C}$. If a node had both this information and for example the path $\mathrm{B_G G} \cdot \mathrm{C_B B_C} \cdot \mathrm{DC_D}$, it could see that the $\mathrm{C_B B_C}$ block is common between the two and find out that B and G are friends because successive links always have a common node, B in this case. If instead the two requirements stated above are satisfied, no attack is possible as we will show in Section 4.2.

Hiding path length. As a further protection measure, PTAPS also has the ability to hide the length of an encrypted path. Specifically, a node N may use additional encrypted blocks, $\mathrm{N_N}$, containing its own identifier (and the random noise). Whenever N writes the identity of a node F either as a successor or as a predecessor in a path, it writes the $\mathrm{F_N}$ block followed by a random number of



(a) Step 1

(b) Step 2

Figure 2: Building routing information.

copies of the $\mathrm{N_N}$ block (each with a different noise sub-block), thus making it impossible for another node to guess the length of a path.

Due to path reversal, a sequence of blocks to be decrypted may have either of two forms: $\mathrm{F_N}, \mathrm{N_N}, ..., \mathrm{N_N}$, or $\mathrm{N_N}, ..., \mathrm{N_N}, \mathrm{F_N}$. Moreover it can be followed either by a block $\mathrm{F_N}$ encrypted by N or by a block encrypted by a different node. To decrypt it, N continues to decrypt blocks until it has found a block of type $\mathrm{F_N}$ with $F \neq N$ and has reached either a block encrypted by a different node, or a block of type $\mathrm{S_N}$ with $S \neq N, F$. In the latter case, it leaves $\mathrm{S_N}$ in the path for later decryption. To simplify the presentation of the protocol, in the rest of the paper, we ignore the use of $\mathrm{N_N}$ blocks to hide path length. Their integration in the descriptions and pseudocode that follow is conceptually simple and is thus left to the reader.

Path-encryption details. Algorithms 1 to 3 provide the details of the path-encryption protocol. Each node, N, maintains three additional fields for each entry, F, in its explicit view. The first is an *outgoing public link* to F, $Pub_{out} = \mathrm{FN_F}$, which will be used to construct encrypted paths with F as an endpoint. The second is an *incoming public link* to N, $Pub_{in} = \mathrm{F_N N}$, which will be used in paths that have N as an endpoint. Finally, the last field is a *secret link*, $Secret = \mathrm{F_N N_F}$, which will be used for paths in which neither N nor F is an endpoint.

To initialize these fields, N sends an UPDATE message to F carrying two blocks containing the identity of F. The first *secret block*, $\mathrm{F_N}$, will be used to create the *secret link* between N and F. The second one, $\mathrm{F'_N}$, will be used to build the *public* link referring to N. Upon receiving these blocks, F also

```
(01) To send an application message M to X through the path P_X
(02)     NextBlock ← GetFirstBlock(P_X)
(03)     NextHop ← ExtractIdentity(NextBlock)
(04)     P'_X ← RemoveFirstBlock(P_X)
(05)     send AppMsg(M, P'_X) to NextHop.

(06) When AppMsg(M, P_X) received from Sender
(07)     PreviousBlock ← GetFirstBlock(P_X)
(08)     PreviousHop ← ExtractIdentity(PreviousBlock)
(09)     if Sender ≠ PreviousHop
(10)         return
(11)     P'_X ← RemoveFirstBlock(P_X)
(12)     if P'_X = ∅
(13)         deliver M
(14)     else
(15)         NextBlock ← GetFirstBlock(P'_X)
(16)         NextHop ← ExtractIdentity(NextBlock)
(17)         P''_X ← RemoveFirstBlock(P'_X)
(18)         send AppMsg(M, P''_X) to NextHop.
```

Algorithm 3: Sending and Relaying Application Messages.

generates a pair N_F and N'_F and sends it back to N. Then the two nodes have the material to create the three encrypted links needed by the protocol, as described in Algorithm 1.

After initialization, nodes operate as in the *TAPS2* protocol when exchanging information from their respective TAPS views. However, some special actions are needed depending on the type of exchanges outlined in Sections 3.3 and 4.1.1. The first type of exchanges, EXPLICIT-EXPLICIT, involve a node that sends information about an explicit friend to another explicit friend. Consider node C in Figure 2a sending information about its friend F to its other friend D. First, C sends D the *outgoing public link* in its routing-table entry for F, FC_F (lines 2 and 3, in Algorithm 2). Upon receiving this link, D concatenates it with the *incoming public link* in its routing table entry for C , $\text{C}_\text{D}\text{D}$. This yields the path $\text{A}_\text{B}\text{B} \cdot \text{FA}_\text{F}$ (lines 17-19 in Algorithm 2).

The second type of exchanges, TAPS-EXPLICIT, involve a node that sends a path leading to a TAPS-view node to an explicit friend. As an example, consider again node C sending a path leading to A to its friend D. Also, let B be the intermediate node in this path: $\text{B}_\text{C}\text{C} \cdot \text{AB}_\text{A}$. Node C cannot send this path to D as it is. If it were to do so, D would combine it with an *incoming public link*, $\text{C}_\text{D}\text{D}$, and would break privacy, since the path $\text{C}_\text{D}\text{D} \cdot \text{B}_\text{C}\text{C} \cdot \text{AB}_\text{A}$ contains node C both in a secret and in a public block, disclosing the fact that C_D refers to C. Thus, C sends a modified path, where it replaces the *public* link $\text{B}_\text{C}\text{C}$ by the secret one, $\text{B}_\text{C}\text{C}_\text{B}$, yielding $\text{B}_\text{C}\text{C}_\text{B} \cdot \text{AB}_\text{A}$. Node D can then concatenate this path to $\text{C}_\text{D}\text{D}$, yielding $\text{C}_\text{D}\text{D} \cdot \text{B}_\text{C}\text{C}_\text{B} \cdot \text{AB}_\text{A}$, which does not pose any privacy problem.

The final type of exchanges are those in which a node N sends information to a node from its TAPS view. As explained in Section 4.1.1, *PTAPS* forbids EXPLICIT-TAPS exchanges. Therefore, the only content of an interaction with a

node from the TAPS view is always information extracted itself from the TAPS view. These are easier to handle and require no special processing since paths contained in the TAPS view (and also in the TAC one) all start and end with a *public link*. This means that they can be concatenated without any special actions. As an example, consider again Figure 2a. This time, node C wishes to inform node E about a path to A. As in the standard *TAPS* [1] and *TAPS2* protocols, C provides E with a path $C \rightarrow A$ ($\mathrm{B_C C \cdot AB_A}$) as well as with a $C \rightarrow E$ ($\mathrm{D_C C \cdot ED_E}$) path. Node E can then combine $C \rightarrow A$ with the best path to C it has, either one it already had or the reverse of the $C \rightarrow E$ it just received. In the latter case, E simply takes the blocks of the received path in reverse order yielding $\mathrm{D_E E \cdot CD_C}$. It then combines this $E \rightarrow C$ path with the $C \rightarrow A$ path it received, yielding $\mathrm{D_E E \cdot CD_C B_C C \cdot AB_A}$.

Encrypted paths clearly allow routing of application messages. These include the verification message described in Section 2.2.3, or messages employed to establish transactions between nodes. The routing algorithm is given in Algorithm 3. Consider node F in Figure 2b wishing to send a message along its path to node D. Node F picks the first block of the path and extract the identity of the node it contains (line 03). Since the identity of a node is either encrypted or in clear form, extracting it from a block means either decrypting it or simply returning the clear-text value in the block. The identity extracted by F indicates that A is the next node on the path, thus F forwards the message to A along with the path stripped of the first block. Upon reception, node A reads the first block of the received path (line 08), which confirms that F is a previous hop. Then, because the path contains more blocks, A proceeds by extracting its next hop (line 16). The process goes on until the message reaches D, which reads the last block of the path on line 08, confirming node C as a previous hop, and then delivers the message at line 13.

Finally it is worth observing that maintaining routing tables at each node with a predecessor and a successor along each path would quickly lead to a waste of storage resources. *TAPS* exchanges lead to the creation of a huge number of paths, many of which are simply discarded when better paths are found. While stale paths could, in principle, be garbage collected, this would require expensive coordination among a large number of distant nodes, and would become almost impossible in the presence of churn.

4.1.3. Strong Threshold

In addition to encrypting paths and restricting the dissemination of information about the trust values on explicit links, *PTAPS* also reinforces the trust threshold we introduced in Section 3.2. Specifically, *TAPS2* removes from a node's view all the nodes whose trust values are below the threshold, but it allows the node to exchange information with them, possibly providing them with trust information about itself and other nodes. *PTAPS*, on the other hand, also forbids such exchanges by ignoring all the communication arriving from nodes whose trust value is below the threshold and by suppressing all outgoing communication towards the same nodes. While this additional measure is not necessary to prevent nodes from discovering trust values on explicit links, it

contributes to limit the ability of attackers to build even an approximate map of the network's trust values by enventually confining nodes' communication to the part of the network where there are sufficiently trusted. Finally, it is important to observe that in *PTAPS*, the trust threshold becomes the primary mechanism for limiting the length of trusted paths, due to the impossibility to perform on-line shortcuts in the presence of encrypted path information.

4.2. Privacy preservation in PTAPS

We now analyze *PTAPS*'s ability to provide strong privacy guarantees by hiding the direct trust values between explicit friends. Intuitively, this ability is provided by the non-disclosure of explicit friends discussed in Section 4.1.1. In the following, we formalize it by clearly specifying the hypothesis under which *PTAPS*'s strong guarantees hold.

Extracting information from a single path. We begin by showing that an attacker cannot extract information about direct links between nodes by examining a single path.

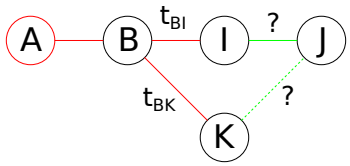


Figure 3: If $t_{BK} \geq t_{BI} \cdot t_{IJ}$, A can't guess if the path goes through I or K.

Lemma 1. *In the routing information provided during path exchanges, if a node's information appears in clear form, then the previous and next nodes (if any) have their routing information hidden.*

Proof. The proof relies on the following observations:

- EXPLICIT-EXPLICIT exchanges, which constitute the bootstrap process, ensure that the routing information associated with paths of length two hides the intermediary node;
- TAPS-EXPLICIT exchanges ensure that the identities of the second and second-to-last nodes in a path remain hidden after the exchange;
- TAPS-TAPS exchanges, which comprise path-reversal and concatenation operations, also preserve the property of the lemma because the endpoints of the paths being concatenated are always represented by public blocks.

As a result, for every path, the identity of at least one node every two remains hidden during all exchanges, thereby proving the lemma.

□*Lemma 1*

Corollary 1. *An attacker cannot guess the existence of a direct link in the explicit social network from the routing information contained in a path.*

Remark: The identities encrypted by the attacker itself in paths are those of its friends. Consequently, even if the attacker can read this information, it can only discover links that join one of its friends to another node, but these links are already disclosed during EXPLICIT-EXPLICIT exchanges.

Extracting information from multiple paths. Next, we present a set of results that show that an attacker cannot extract information about direct links by trying to identify common path subsequences in two paths. These results cover cases in which an attacker gathers a finite number of paths and combines them in pairs by trying to identify that one path is a subpath of the other. However, they do not cover situations in which an attacker can gather some form of exhaustive knowledge about the paths in the network: for example by being sure to have collected all the two-hop paths starting from a node it wishes to attack.

It is easy to see that the only combination of two paths that might lead to the identification of a direct link consists of a path, p_1, and a path p_2 corresponding to p_1 with the addition of a final node J. Lemmas 2 and 3 prove that even this combination cannot leak friendship information. Their proofs, as well as those of the theorem and corollary that follow, rely on a few standard assumptions on the cryptographic algorithms used to encrypt routing blocks.

- It is impossible to tell which key was used to encrypt a block without having the appropriate key.
- It is impossible to tell if two blocks were encrypted with two different keys without being able to decrypt at least one of them.
- It is impossible to tell whether two blocks encrypted with different keys contain the same information.

Lemma 2. *Let* A *be an attacker that has the knowledge of two paths $p_1 = ABI$ and $p_2 = ABIJ$ along with their associated routing information. If neither I nor J is a friend of A, and if B has at least another friend K such that: (i) K is different from A, I, and J, (ii) K is not a friend of A, and (iii) $t_{BK} \geq t_{BI} \cdot t_{IJ}$, then A cannot guess the existence of link $I - J$.*

Proof. Because of the behavior of EXPLICIT-EXPLICIT exchanges, the routing information of path p_1 is $\text{B'}_{\text{A}}\text{A} \cdot \text{IB'}_{\text{I}}$, while that for path p_2 is $\text{B'}_{\text{A}}\text{A} \cdot \text{I}_{\text{B}}\text{B}_{\text{I}} \cdot \text{JB'}_{\text{J}}$. For A , it is therefore impossible to detect that IB'_{I} and $\text{I}_{\text{B}}\text{B}_{\text{I}}$ refer to the same link. First A does not have B 's key and is thus unable to match block I_{B} with block I. Second, she does not have I 's key either, and thus the routing blocks B_{I} and B'_{I} also look different to her. This means that because of the hypothesis that B has another friend $\text{K} \neq \text{A}, \text{I}, \text{J}$, A cannot distinguish the routing information for p_2 from that associated with a path $p'_2 = \text{ABKJ}$. Moreover, because A is not a friend of K, she cannot exclude the existence of link $\text{K} - \text{J}$. As a result, the only way for A to establish that p_2 is not p'_2 would be to rely on trust values, and realize that the trust value associated with the link $\text{B} - \text{K}$ is too low for p'_2 to have the same trust value as p_2. However, this would require the trust value of $\text{B} - \text{K}$ to be $t_{BK} < t_{BI} \cdot t_{IJ}$, which contradicts the hypothesis. This proves that A cannot identify if p_2 goes through I or through K, and thus cannot guess the existence of the link between I and J .

$\square_{Lemma\ 2}$

Lemma 3. *Let A be an attacker that knows two paths $p_1 = B_1 \dots B_n I, n \geq 2$ and $p_2 = B_1 \dots B_n IJ$. If $A \notin \{B_1, \dots, B_n, I, J\}$ and if neither B_n, I nor J is a friend of A, then A cannot guess the existence of link $I - J$.*

Proof. The routing information of p_1 ends with the blocks $\text{IB'}_{n\text{I}}$, while that of p_2 ends either with the blocks $\text{I}_{\text{B}_n}\text{B}_{n\text{I}} \cdot \text{JI'}_{\text{J}}$, or with the blocks $\text{I'}_{\text{B}_n}\text{B}_n \cdot \text{JI'}_{\text{J}}$. We now prove that the attacker is unable to match the two paths because the corresponding blocks in their final links differ. First, I differs from its encrypted counterpart I_{B_n}. Second, $\text{B'}_{n\text{I}}$ differs both from its its unencrypted counterpart B_n and from $\text{B}_{n\text{I}}$. The encrypted block for B_n appearing in public links, $\text{B'}_{n\text{I}}$, is, in fact, different from the one appearing in secret links, $\text{B}_{n\text{I}}$, as described in Section 4.1.2.

In addition A is neither a friend of B_n nor a friend of J, thus it has no way to exclude that path p_2 contains ends, for example, by the sequence $\text{B}_n - \text{K} - \text{J}$. This, proves that she cannot determine whether p_2 contains a link between I and J .

$\square_{Lemma\ 3}$

Remark: The lemma trivially holds with the same arguments if $p_1 = \text{IB}_n \dots \text{B}_1$ and $p_2 = \text{JIB}_n \dots \text{B}_1$.
Lemmas 2 and 3 state the hypotheses required to prevent attacks from a specific attacker. In the following we provide sufficient conditions that are valid regardless of the attacker.

Theorem 1. *Let I and J be two explicit friends such that: (i) neither I nor J is friends with a malicious node, (ii) each friend of I has at least two other friends $\neq J$ which are not friends with each other and that it trusts at least as much as I, and (iii) each friend of J has at least two other friends $\neq I$ which are not friends with each other and that it trusts at least as much as J. Then, no attacker, regardless of its position in the network, can discover with certainty that a given path contains an explicit link between I and J through any combination of two paths.*

Proof. From the hypotheses, it immediately follows that, if the attacker and I (resp. J) have a common friend, then this has at least one friend $\text{K} \neq \text{A}, \text{I}, \text{J}$ which it trusts at least as much as I (resp. J) and which is not friends with A. Thus the hypotheses of Lemmas 2 and 3 are verified for any attacker A.

$\square_{Theorem\ 1}$

Corollary 2. *If every node in the network gives its maximum trust value to at least three of its friends which are not friends with each other, then given any two explicit friends I and J that are not friends with malicious nodes, no attacker can discover with certainty that a given path contains an explicit link between I and J.*

Proof. It is easy to observe that the fact that each node has at least three friends which are not friends with each other and which have the same highest trust values makes it possible to find, for each friend of I's (resp. J's), one friend which is (i) at least as trusted as I (resp. J), (ii) not equal to J (resp. I) or A , and (iii) not friends with A. This verifies the hypotheses of Theorem 1. $\square_{Corollary\ 2}$

The above results precisely describe the circumstances that prevent an attacker from discovering the existence of friendship relationships. Specifically, Lemma 1 and Corollary 1 guarantee that an attacker cannot extract link or trust information from a single path. Lemma 2, Lemma 3, Theorem 1, and Corollary 2 extend this guarantee to an attacker trying to extract information from the combination of two paths.

5. Evaluation

5.1. Setting

We evaluated our protocols on real data traces consisting of 3000 users extracted from the Facebook and Digg social websites. The Facebook trace[3] contains friendship links and a list of social interactions. To obtain a treatable subset for our experiments, we first cleaned up the trace by removing all the users that had only one friend, as they would be too isolated to benefit from our social platform. We then selected the user with the largest number of interactions and proceeded in a spiral fashion by selecting her friends, then the friends of her friends, and so on, until we reached 3000 users. We associated each of these users with a random user profile from the Digg social network. We obtained these profiles by crawling Digg in late 2010.

Friendship links in the Facebook trace and profiles in the Digg trace provided the base of explicit links and profiles for our experiments. On top of them, we built several traces by varying the trust patterns between the nodes. We distinguish our traces into two groups: binary and multi-valued. In both, we make the assumption that the number of interactions between two nodes is a measure of trust (this assumption is not part of the protocol itself).

Our choice of a random mapping between traces is due to the absence, to the best of our knowledge, of a real trace combining profiles and trust information. The random mapping may in fact underestimate performance. *TAPS* [1], *TAPS2*, and *PTAPS* are particularly good at discovering trusted paths toward close nodes in the explicit social network and, as shown in [9], friends in a social network tend to be more similar to each other than to random nodes.

Binary traces. In binary traces we assigned a binary trust value to each link in the data set. Specifically, we sorted the friends of each user according to the number of interactions she had with them. Then, for a user with $|N|$ friends, we assigned a trust value of 1 to the $\beta|N|$ directed links with the largest number of interactions. As this process creates asymmetric trust values, we then set the symmetric trust value of each link as the logical OR of the two asymmetric values. This leads to the proportions of trusted links depicted in Figure 4a.

[3]Network A at `http://current.cs.ucsb.edu/facebook/index.html`.

Multivalued traces. While binary traces provide a simple experimental setting, reality tends to be more complex. Thus, we also considered traces with trust values of 1, 0.8, 0.5, and 0. Similar to the binary case, we sorted each user's friends by the number of interactions and assigned a trust value of 1 to the top $\gamma_1|N|$, 0.8 to the following $\gamma_{0.8}|N|$ and so on, leading to the traces in Figure 4b.

5.2. Terms of Comparison

We compared the performance of our *TAPS* protocols with several alternatives. *Best* is an ideal system that, powered with global knowledge, always provides each user with the set of neighbors that offer the best combination of similarity and trust. This allows us to assess the ability of our *TAPS* variants to reach similar results in a decentralized way. *Similar* consists of a standard similarity-based implicit network [4], augmented with an oracle providing the best trusted paths to neighbors. This maximizes profile similarity at the cost of possibly lower trust. Finally, *Trusted* selects the most trustworthy nodes, providing very high trust, but possibly poor similarity. In all experiments, we compute the average scores of the TAC views over all nodes using each of the *TAPS* variants, as well as *Similar* and *Trusted*. Then, we normalize each of these averages by dividing it by the average score over all nodes obtained by *Best*, which provides optimal views. Unless otherwise specified, we used default values for τ and ϵ: $\tau = 0.75$ balances trust decay and path length (0.56 at 3 hops, 0.23 at 6), and $\epsilon = 1$ gives a fair tradeoff between trust and similarity.

5.3. Results

Impact of trust density. We start our performance comparison by examining the results obtained in the various traces to highlight the effect of the number of trusted links. Figure 5 shows the average score values in the TAC views with *TAPS2* and *PTAPS* as well as with their competitors as percentages of the scores of *Best*. *TAPS2*'s and *PTAPS*'s performances are always better than those obtained by *Similar* and *Trusted* with the use of global knowledge. As expected, they particularly outperform *Similar* whenever the social network has a limited proportion of high-trust links (Binary0.4 and Multi-valued1). This can be explained by observing that *Similar* always selects the nodes with the best similarity and is therefore penalized in networks with lower trust density. On the other hand, *Trusted* performs best with a low trust density, but its performance remains bad in any configuration. We also observe that *PTAPS*'s performance is only slightly below that of *TAPS2*, and that both protocols achieve stable behaviors across all traces with respect to *Best*, contrary to *Similar* and *Trusted*.

Impact of trust transitivity. Figure 6 confirms the above observations by examining the impact of trust transitivity, τ, in the Binary-0.8 and MultiValued-1 traces. The performance of both our protocols is particularly good when trust decays faster as this gives more importance to nodes that are closer in the social network even if they may be less similar. This suggests that our *TAPS* protocols are particularly suited for important transactions and situations in which people can tolerate only limited amounts of risk.

Name	β	% 1	% 0
Binary-0.4	0.4	53.5	46.5
Binary-0.6	0.6	71.3	28.7
Binary-0.8	0.8	89.3	10.7

(a) Binary Traces

Name	% 1	% 0.8	% 0.5	% 0
MultiValued-1	41.3	23.8	23.4	11.5
MultiValued-2	57.4	27.9	13.3	1.4
MultiValued-3	76.2	12.3	10.1	1.4

(b) Multi-valued Traces

Figure 4: Trust values distribution for different traces.

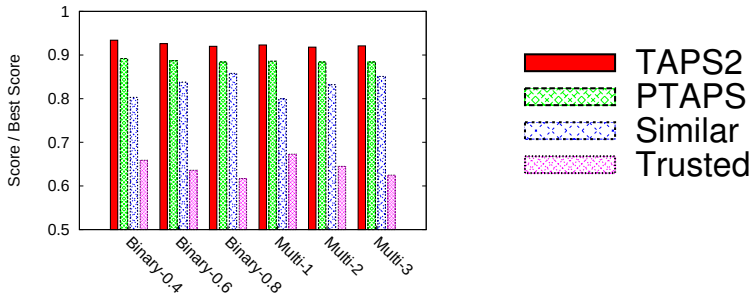
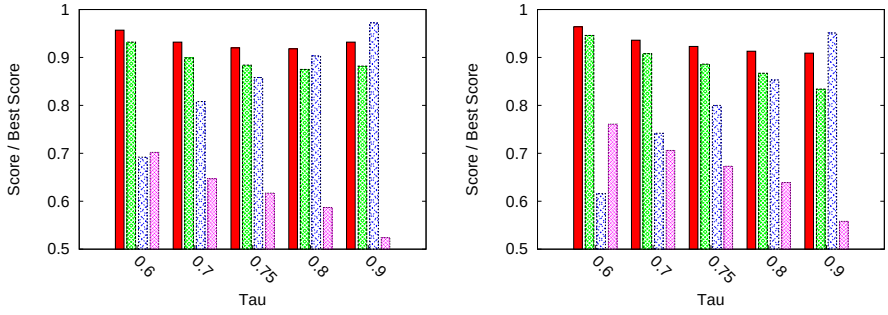


Figure 5: Impact of trust density.

With very high values of τ, trust decays more slowly. In this case, a protocol like *Similar* that selects the most similar nodes before searching for a trusted path may be viable. However, *Similar* achieves this through global knowledge. A distributed protocol to compute trusted paths would probably be either very costly or ultimately equivalent our *TAPS* family in networks characterized by high trust densities. Moreover, such a protocol would remain weak in situations where the density of trusted links is low, as shown in Figure 5.

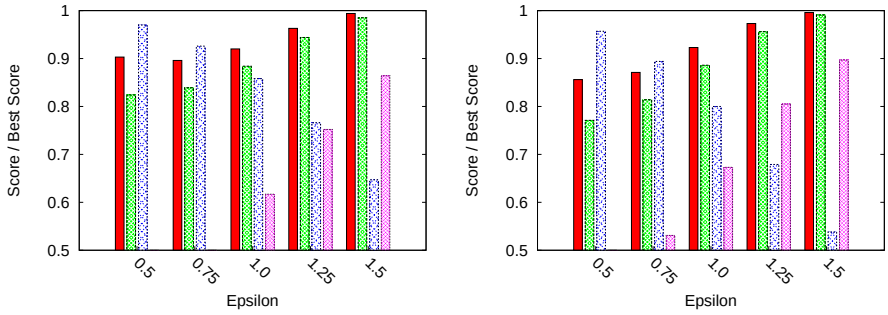
The plots also show that the importance of short-circuiting cyclic paths is greater when τ is high. High τ values make it possible for the protocols to select nodes that are farther away in the social network, which, in turn, makes the presence of cycles more likely. This effect is mitigated by *TAPS2*'s ability to take shortcuts, but it is clearly visible in *PTAPS*, which does not support on-line shortcuts.

Impact of trust weight. Next, we evaluate the impact of the *trust-weight* factor. Figure 7 shows the results in the Binary-0.8 (left) and Multi-valued-1 (right) traces respectively. Both plots show that the benefits of a protocol like *TAPS2*/*PTAPS* become more important as we place more weight on trust. With values of 1 or above, *TAPS2* and *PTAPS* perform better than *Similar* and reach almost optimal results with $\epsilon = 1.5$, while *Trusted* is only able to achieve acceptable results with a very high value of ϵ. Similar to Figure 6, Figure 7 also highlights that the benefits of shortcuts are greater when the similarity between nodes is more important than trust. The difference in performance between *TAPS2* and *PTAPS* decreases as the impact of trust increases.



(a) Binary. (b) Multi-valued.

Figure 6: Impact of τ in the Binary0.8 (left) and Multi-valued-1 (right) traces.



(a) Binary. (b) Multi-valued.

Figure 7: Impact of ϵ in the Binary0.8 (left) and Multi-valued1 (right) traces.

Impact of the biased view and the trust threshold. We now concentrate on the components of our protocols and analyze the impact of biased views and of the trust threshold on *TAPS2* and *PTAPS* on the MultiValued-2 trace. Figure 8 highlights that biasing view selection significantly improves the performance of both *TAPS2* and *PTAPS* by directing the exploration of the network toward highly trusted links. The trust threshold complements this by obtaining an additional increment in performance. The presence of the trust threshold becomes even more critical if we disable the biased-view-selection feature. The threshold alone, even if not as effectively as when biasing view selection, prevents non-trustworthy nodes from entering a node's view.

Comparison with TAPS [1]. Figure 8 also shows a comparison between *TAPS2*, *PTAPS* and *TAPS* [1]. It is clear that *TAPS2* strongly outperforms *TAPS* [1]. Even in its simplest version, without the biased view and the trust threshold, *TAPS2* remains slightly better than *TAPS* [1] thanks to the additional exchanges. *PTAPS* trades off most of the performance improvements of *TAPS2*

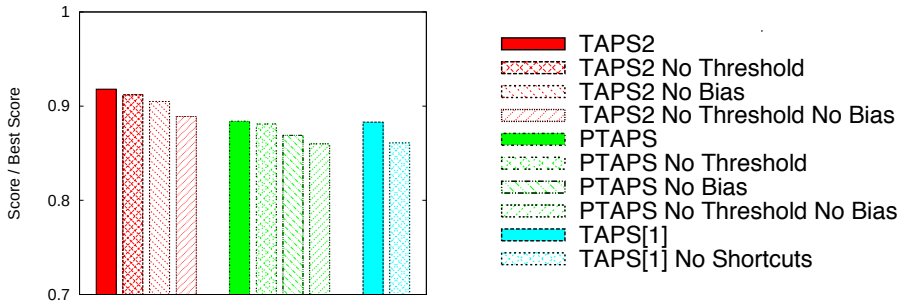
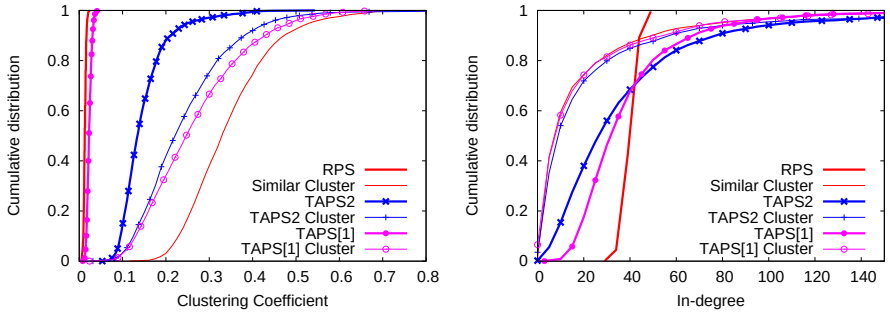


Figure 8: Impact of view bias and trust threshold.



(a) Clustering coefficient distribution.

(b) In-degree distribution.

Figure 9: Cumulative distributions of the local clustering coefficients (left) and of the in-degree (right) distributions for *TAPS*-based and standard protocols.

for its privacy guarantees. Yet, it remains slightly better than *TAPS* [1] when all its features are enabled even though it does not use the shortcut mechanism.

Graph properties of TAPS. We conclude our evaluation by examining the graph properties of our *TAPS* overlays. We do not show *PTAPS* because its results are almost indistinguishable from those of *TAPS2*. Figure 9a shows the cumulative distributions of the local clustering coefficients of the nodes in the TAPS and TAC views for both *TAPS* [1] and *TAPS2* and compares them with those of a standard peer sampling protocol coupled with a clustering protocol (RPS and *Similar* Cluster). The plot shows that *TAPS* [1]'s topology is very close to a random one, while *TAPS2*'s is clearly more clustered. This is a clear effect of *TAPS2*'s biased view selection which tends to favor more trustworthy nodes and suggests the use of *TAPS2*/*PTAPS* for better performance/privacy and of *TAPS* [1] if sharing the overlay with non-trust-aware applications. The plot also shows that the TAC views generated by both *TAPS2* and *TAPS* [1] are less dense than those of *Similar* as very similar nodes that form tight clusters in *Similar* may not be connected by highly trusted links.

The in-degree distribution shown in Figure 9b also shows some differences with respect to traditional protocols. In this case, *TAPS2* and *TAPS* [1] produce very similar results, but are quite different from the RPS views. The in-degree distribution of TAPS views is in fact slightly skewed because nodes that are not trusted by many others tend to have fewer neighbors. This is indeed desirable as untrusted nodes could potentially harm the system through illicit behaviors. Finally, we observe that the indegree distribution for the TAC views does not show any significant variation among *TAPS2*, *TAPS* [1], and *Similar*.

6. Related Work

The concept of trust in explicit social networks has been exploited in domains ranging from peer-to-peer security to recommendation systems. SybilGuard [10] and SybilLimit [11] propose protocols that exploit trust relationships between friends to protect peer-to-peer systems from sybil attacks. Reliable Email [12] uses a similar approach to build an email-whitelisting system based on friend-to-friend relationships, while Ostra [13] exploits social trust to limit the incidence of unwanted communication in messaging and content-sharing systems.

NABT [14] proposes the use of trust between friends to prevent freeriding behaviors using a more efficient form of tit-for-tat based on indirect trust relationships. NABT's credit-based approach can be viewed as a basic form of trust inference between friends of friends. A more advanced approach is adopted by SUNNY [15], a centralized protocol that takes into account both trust and confidence to build a Bayesian network. Even if SUNNY is centralized, its confidence-based idea could lead to interesting improvements for *TAPS*.

A number of research efforts have instead investigated the use of trust links to improve the performance of recommendation systems. The work in [16] uses an approach similar to that of [15], while TrustWalker [17] combines trust and item-based collaborative filtering. TaRS [18] builds a recommendation system capable of operating both with global and with local trust metrics. Global trust metrics [19] predict a global reputation value for each node. Local trust metrics [20], on the other hand, take an approach similar to ours and compute trust values that are dependent on the target user.

Despite the mole of work on social trust, Social Market is, to the best of our knowledge, the first system to propose the use of trust relationships to build a decentralized interest-based marketplace. Similarly, *TAPS*, originally published in [1], *TAPS2*, and *PTAPS* constitute the first protocols to combine explicit and implicit social networks into a single gossip-based system. Existing research on gossip protocols has addressed a number of problems including data dissemination [21], aggregation [22], and overlay construction and maintenance [23, 8]. In this context, the two contributions that are most closely related to our work are [24], which uses gossip to disseminate news in explicit networks, and [4], which proposed the use of gossip for implicit ones.

Finally, our path-encryption mechanism shares similar aspects with onion-routing approaches [25]. In our protocol however, the endpoints of a path need

to be able to reverse paths, and, most importantly, they must not know the identities of the nodes in a path, making the use of [25] impossible.

7. Conclusions

We presented Social Market (SM), a novel distributed application enabling trusted collaborative actions between similar people in a social-network environment. We proposed a solution to the challenges posed by SM with a family of novel protocols, *TAPS*, *TAPS2*, and *PTAPS*, which create RPS-like overlays taking into account the mutual trust expressed by users when joining an explicit social network. *TAPS2* and *PTAPS* provide significant improvements with respect to our previous work on *TAPS* both in terms of performance and privacy. While *TAPS2* focuses on finding nodes with the best trust values, *PTAPS* also provides strong privacy guarantees by hiding the existence of explicit links from nodes that are at more than two hops from them.

Our results encourage us to extend Social Market and the *TAPS* family in a number of ways. First, we are currently evaluating whether the protection offered by *PTAPS* can be extended to combinations of more than two paths. Second, we are working on extensions capable of addressing situations in which attackers may inject false information with the aim of polluting other nodes' views or of detecting the presence of links between people. A third direction we are considering is to extend our protocol family to networks characterized by asymmetric trust values as opposed to symmetric ones. This would make it possible to render trust information even more private as users would not need to disclose to their friends the trust they have for them. Both with symmetric and asymmetric trust values, it would also be interesting to explore the use of multiple redundant trusted paths as a way to limit the effects of colluding attackers, which cannot be tolerated by the protocols in this paper. From a systems perspective, we instead plan to evaluate and possibly improve the performance of our protocols in the presence of churn, for example by having nodes rely on trusted peers to disseminate information while disconnected. Finally, we aim to integrate *TAPS* in our existing prototype applications within the Gossple project as a way to reinforce the trust of users in disseminated information, recommendations, and ultimately in the implementation of a real-world social-market platform.

Acknowledgments. This work is supported by the ERC Starting Grant GOSSPLE number 204742. We are grateful to Guang Tan for his initial contributions, as well as to the SSS-2011 reviewers for their constructive suggestions.

References

[1] D. Frey, A. Jégou, A.-M. Kermarrec, Social market: combining explicit and implicit social networks, in: 13th international conference on Stabilization, safety, and security of distributed systems (SSS'11), pp. 193–207.

[2] M. Bender, T. Crecelius, M. Kacimi, S. Miche, J. Xavier Parreira, G. Weikum, Peer-to-peer information search: Semantic, social, or spiritual?, Data Engineering Bulletin 30 (2007) 51–60.

[3] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, H. Jeong, Analysis of topological characteristics of huge online social networking services, in: 16th international conference on World Wide Web (WWW '07), New York, NY, USA, pp. 835–844.

[4] M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, V. Leroy, The gossple anonymous social network, in: International Conference on Middleware (Middleware '10), pp. 191–211.

[5] G. Salton, M. J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, Inc., 1986.

[6] X. Bai, Personalized top-k processing: from centralized to decentralized systems, Ph.D. thesis, INSA Rennes, France, 2010.

[7] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, M. van Steen, Gossip-based peer sampling, ACM Trans. Comput. Syst. 25 (2007).

[8] S. Voulgaris, M. van Steen, Epidemic-style management of semantic overlays for content-based searching, in: 11th International Euro-Par Conference on Parallel Processing (Euro-Par'05), pp. 1143–1152.

[9] M. McPherson, L. S. Lovin, J. M. Cook, Birds of a Feather: Homophily in Social Networks, Annual Review of Sociology 27 (2001) 415–444.

[10] H. Yu, M. Kaminsky, P. B. Gibbons, A. D. Flaxman, Sybilguard: defending against sybil attacks via social networks, IEEE/ACM Transactions on networking 16 (2008) 267–278.

[11] H. Yu, P. B. Gibbons, M. Kaminsky, F. Xiao, Sybillimit: a near-optimal social network defense against sybil attacks, IEEE/ACM Trans. Netw. 18 (2010) 885–898.

[12] S. Garriss, M. Kaminsky, M. J. Freedman, B. Karp, D. Mazières, H. Yu, Re: reliable email, in: 3rd conference on Networked Systems Design & Implementation - Volume 3 (NSDI'06), pp. 22–22.

[13] A. Mislove, A. Post, P. Druschel, K. P. Gummadi, Ostra: leveraging trust to thwart unwanted communication, in: 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08), pp. 15–30.

[14] Z. Liu, H. Hu, Y. Liu, K. W. Ross, Y. Wang, M. Mobius, P2p trading in social networks: the value of staying connected, in: Proceedings of the 29th conference on Information communications (INFOCOM '10), Piscataway, NJ, USA, pp. 2489–2497.

[15] U. Kuter, J. Golbeck, Sunny: a new algorithm for trust inference in social networks using probabilistic confidence models, in: 22nd national conference on Artificial intelligence - Volume 2 (AAAI'07), pp. 1377–1382.

[16] Y. Wang, J. Vassileva, Bayesian network trust model in peer-to-peer networks, in: Second International Workshop Peers and Peer-to-Peer Computing (AP2PC '03), pp. 23–34.

[17] M. Jamali, M. Ester, Trustwalker: a random walk model for combining trust-based and item-based recommendation, in: 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KKD'09), KDD '09, pp. 397–406.

[18] P. Massa, P. Avesani, Trust-aware recommender systems, in: 2007 ACM conference on Recommender systems (RecSys '07), pp. 17–24.

[19] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Computer Networks and ISDN Systems 30 (1998) 107–117.

[20] P. Massa, P. Avesani, Controversial users demand local trust metrics: an experimental study on epinions.com community, in: 20th national conference on Artificial intelligence - Volume 1 (AAAI'05), pp. 121–126.

[21] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, Y. Minsky, Bimodal multicast, ACM Transactions on Computer Systems 17 (1999) 41–88.

[22] M. Jelasity, A. Montresor, O. Babaoglu, Gossip-based aggregation in large dynamic networks, ACM Transactions on Computer Systems 23 (2005) 219–252.

[23] M. Jelasity, O. Babaoglu, T-man: gossip-based overlay topology management, in: 3rd international conference on Engineering Self-Organising Systems (ESOA'05), pp. 1–15.

[24] A. Datta, R. Sharma, Godisco: selective gossip based dissemination of information in social community based overlays, in: 12th international conference on Distributed computing and networking (ICDCN'11), pp. 227–238.

[25] M. Reed, P. Syverson, D. Goldschlag, Anonymous connections and onion routing, IEEE Journal on Selected Areas in Communications 16 (1998) 482–494.